Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 2468 – 2471

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Research on Scheduling Scheme for Hadoop clusters

Jiong Xie[a,b], FanJun Meng[c], HaiLong Wang[c], HongFang Pan[b], JinHong Cheng[b], Xiao Qin[a]

[a]*Inner Mongolia Electric Power Information and Communication Center, Hohhot, China*
[b]*Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849-5347, USA*
[c]*Computer & Information Engineering College, Inner Mongolia Normal University, Hohhot, China*

Abstract

In this paper, we import a prefetching mechanism into MapReduce model while retaining compatibility with the native Hadoop. Given a data-intensive application running on a Hadoop MapReduce cluster, our approach estimates the execution time of each task and adaptively preloads an amount of data to the memory before the new task is assigned to the computing node. We implement a predictive schedule and prefetching (PSP) mechanism, which is integrated into the native MapReduce runtime system. We also evaluate performance on a 10-node cluster using two popular benchmarks–grep and wordcount. The PSP mechanism reduces the execution time of grep and wordcount up to 28 % with an average of 19%. Moreover, the PSP model increases the overall throughput and improves the I/O utilization. Because of the limitation of length, we did not present the experiment result detail in this paper.

Keywords: MapReduce; Prefetching; Schedule.

1. introduction

In the past decade, the MapReduce framework has been employed to develop a wide variety of data-intensive applications in large-scale systems. In this paper, we observe the data movement and task process pattern of MapReduce, and design a prefetching mechanism to solve this problem so as to improve the performance.

Four factors make predictive scheduling and prefetching desirable and possible:

1. the underutilization of CPU processes
2. the growing importance of MapReduce performance
3. the Hadoop file distribution model offer data storage information
4. interaction between master node with slave nodes.

Our preliminary results [1] show that CPU and IO workload are underutilized while they are working on a data-intensive application. In MaReduce model, HDFS(Hadoop Distributed File System) is tuned to support large files, typically file sizes in HDFS range from gigabytes to terabytes in size. HDFS spilt the big files to several partition and distribute to hundreds of nodes in a single cluster, The HDFS center store the index information, called meta data to manage several partition. These partition is the the basic data element in HDFS, the size of which by

*Jiong Xie. Tel.: +86-186-7247-4088.
E-mail address: JZX0009@auburn.edu.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.423

default is 64MB. The big block size can hidden the seeking latencies. However, because of the big block size, the transfer time of data access dominate the whole process time, and the stall for IO is also a significant factor in the process time. This suggests that prefetching are needed to boost IO performance and improve the performance.

The second factor encourage our prefetching mechanism is that ever-faster CPUs are processing data more quickly than the data provided. Simply increasing the cache does not improving the IO-subsystem performance and CPU performance [2]. In the MapReduce model, before a computing node launches a new task, it should ask the master node to assign one, and then the master node let the computing node know what the next task is and where the required data is located. The computing node does not gather the required data and process until receiving these information. In this way, the CPU should wait a long period while the computing node communicate with the master node. Prefetching strategies are need to parallelize these workloads so as to avoid the idle point.

1.1. Contribution

In this paper, we present a predictive scheduling and prefetching mechanism that improving the performance of MapReduce. We propose the predictive scheduling algorithm to arrange task. Prefetching manages the data loading. The basic idea is to preload the data from local disk to cache as late as possible without any delaying on the new task's launching.

The novelty of this study lies in our new mechanism that integrates a prefetching scheme with a predictive scheduling algorithm. The original Hadoop system randomly assigns tasks to computing nodes and loads data from local or remote disks whenever the data sets are required. CPUs of the computing nodes will not process new tasks until all the input data resources are loaded into the nodes' main memory. The coordination between CPUs and disks in terms of data I/O has a negative impact on Hadoop's performance. In the design of our mechanism, we change the order of the processing procedure, our prefetching scheme assists Hadoop clusters to preload required input data prior to launching tasks on DataNodes. Our algorithms aim at

1. preloading the data from local disk a little early before the new task assigned
2. shortening the waiting period, the new task can be processed in time
3. improving the performance of MapReduce system

We have evaluated our solutions using different benchmarks on Hadoop system. Evaluation results show that our prefetching mechanism can achieve at least 10% reduction in execution time comparing with the original one.

2. Design and implementation

2.1. Predictive Scheduler

We design a predictive scheduler - a flexible task scheduler - to predict the most appropriate task trackers to which future tasks should be assigned. Once the scheduling decisions are predicted ahead of time, DataNodes can immediately start loading $< key, value >$ pairs. Our predictive scheduler allows DataNodes to explore the underutilized disk bandwidth by preloading $< key, value >$ pairs.

Let us start describing this scheudling mechanism by introducing the native Hadoop scheduler. The job tracker includes a task scheduler module to assign tasks to different task trackers. The task tracker periodically sends a heartbeat to the job tracker. The job tracker checks heartbeat and assigns tasks to available task trackers. The scheduler assigns each task to a node randomly via the same heartbeat message protocol. The algorithm for predicting stragglers in the native Hadoop is inadequate, because the original algorithm uses a single heuristic variable for prediction purpose. The native Hadoop randomly assigns tasks and mispredicts stragglers in many cases.

To address the aforementioned problem, we develop a predictive scheduler by designing a prediction algorithm integrated with the native Haddop. Our predictive scheduler seeks stragglers and predicts candidate data blocks. The prediction results on the expected data are sent to corresponding tasks. The prediction decisions are made by a prediction module during the prefetching stage.

We seamlessly integrate the predictive scheduler with the prefetching module. Below let us describe the structure of the prefetching module, which consists of a single prefetching manager and multiple worker threads. The role of the prefetching manager is to monitor the status of worker threads and to coordinate the prefetching

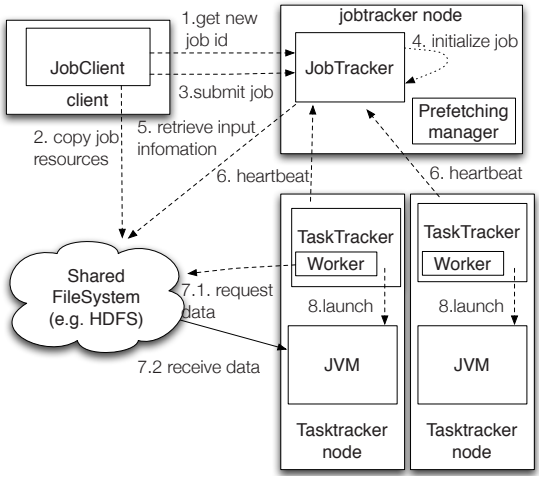

Fig. 1. Three basic steps to launch a task in Hadoop.

process with tasks to be scheduled. When the job tracker receives a job request from a Hadoop application, the job tracker places the job in an internal queue and initializes the job [3][4]. The job tracker divides a large input file to several fixed-size blocks and creates one map task for each block. Thus, the job tracker partitions the job into multiple tasks to be processed by task trackers. When the job tracker receives a heartbeat message from an idle task tracker, the job tracker retrieves a task from the queue and assigns the task to the idle task tracker. After the task tracker obtains the task from the job tracker, the task is running on the task tracker.

Figure 1 shows that the Hadoop system applies the following three basic steps to launch a task. First, the job tracker localizes the job JAR by copying the job from the shared file system to the task tracker's file system. The job tracker also copies any required files by the Hadoop application from the distributed cache to the local disk. Second, the job tracker creates a local working directory for the task, and un-jars the contents of the JAR into this directory. Last, an instance of TaskRunner is created to launch a new Java Virtual Machine to run the task.

In our design, the above task launching procedure is monitored by the prediction module. Specifically, the prediction module in the scheduler predicts the following events. 1) finish times of tasks currently running on nodes of a Hadoop cluster; 2) pending tasks to be assigned to task trackers; and 3) launch times of the pending tasks.

2.2. Prefetching

Upon the arrival of a request from the Job tracker, the predictive scheduler triggers the prefetching module that forces preload worker threads to start loading data to main memory. The following three issues must be addressed in the prefetching module.

When to prefetch. The prefetching module controls how early to trigger prefetching actions. Before a block finishes, the subsequent block will be loaded into the main memory of the node. The prediction module assists the prefetching module to estimate the execution time of processing each block in a node. Please note that the block processing time of an application on different nodes may vary in a heterogeneous cluster. The estimates are calculated by statistically measuring the processing times of blocks on all the nodes in a cluster. This statistic measuring can be performed offline.

What to prefech. The prefetching module must determine blocks to be prefetched. Initially, the predictive scheduler assigns two tasks to each task tracker in a node. When the prefetching module is triggered, it proactively contacts the job tracker to seek required information regarding data to be processed by subsequent tasks.

How much to prefetch. When the prefetching action is triggered, the prefetching module automatically fetches data form disks. Due to the large block size in HDFS, we intend not to make our prefetching module very aggressive. Thus, there is only one block being prefetched at a time.

The most important part of the prefetching work is to synchronize two resources in the MapReduce system: the computing task and the data block. The scheduler in the MapReduce always collects all the running task information and constructs a RunningTaskList. It separately caches the different types of tasks in a map task list and a reduce task list. The job tracker can manage the current task according to these lists [5]. The prefetching manager in the master node constructs a list known as the data list, a collection of all the data block location information. The worker thread in each running node can finish the data loading job all by itself before the task is received.

3. Related work

Prefetching mechanism An array of prefetching techniques have been proposed to improve the performance of main memory in computing systems [6]. Cache prefetching techniques used to improve effectiveness of cache-memory systems have been widely explored for a variety of hardware and software platforms. An increasing number of computing systems are built to support multimedia applications. A few studies were focused on prefetching approaches to boosting I/O performance in computer systems .

Scheduling Algorithms Performance of Hadoop systems can be improved by efficiently scheduling tasks on Hadoop clusters. Many scheduling algorithms might be adopted in Hadoop. For example, the FairScheduler and Capacity Scheduler provide more opportunity for later jobs to get scheduled. Zaharia *et al.* [7] implemented a new scheduling algorithm called LATE (i.e., Longest Approximation Time to End) in Hadoop to improve Hadoop system performance by speculatively executing tasks that decrease response time the most.

4. conclusion

In this paper, we observed that the task processing procedure in Hadoop may introduce data transfer overhead in a cluster. Realizing that the data transfer overhead is caused by the data locality problem in Hadoop, we proposed a predictive scheduling and prefetching mechanism or PSP for short to hide data transfer overhead. Our PSP mechanism seamlessly integrate a prefetching module and a prediction module with the Hadoop's job scheduler. The prediction module proactively predicts subsequent blocks to be accessed by computing nodes in a cluster; whereas the prefetching module preloads these future blocks in the cache of the nodes. The proposed PSP is able to avoid I/O stalls incurred by predicting and prefetching data blocks to be accessed in the future.

Acknowledgments

This research was supported by the U.S. National Science Foundation under Grants CCF-0845257 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502(CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341(CCLI), and DUE-0830831 (SFS), as well as Auburn University under a startup grant, and a gift (No. 2005-04-070) from the Intel Corporation. This research was also supported by Scientific Research Project of Higher Education of Inner Mongolia Autonomous Region,China (NJZY13052)

References

[1] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, X. Qin, Improving mapreduce performance through data placement in heterogeneous hadoop clusters, in: Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on, April, pp. 1–9. doi:10.1109/IPDPSW.2010.5470880.

[2] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, J. Zelenka, Informed prefetching and caching, SIGOPS Oper. Syst. Rev. 29 (1995) 79–95.

[3] T. White, Hadoop The Definitive Guide, O'Reilly, 2009.

[4] J. Venner, Pro Hadoop, Apress, 2009.

[5] S. Seo, I. Jang, K. Woo, I. Kim, J.-S. Kim, S. Maeng, Hpmr: Prefetching and pre-shuffling in shared mapreduce computation environment, in: Proceedings of 11th IEEE International Conference on Cluster Computing, ACM, 2009, pp. 16–20.

[6] A. J. Smith, Cache memories, ACM Comput. Surv. 14 (1982) 473–530.

[7] M.Zaharia, A.Konwinski, A.Joseph, Y.zatz, I.Stoica, Improving mapreduce performance in heterogeneous environments, In OSDI'08: 8th USENIX Symposium on Operating Systems Design and Implementation.